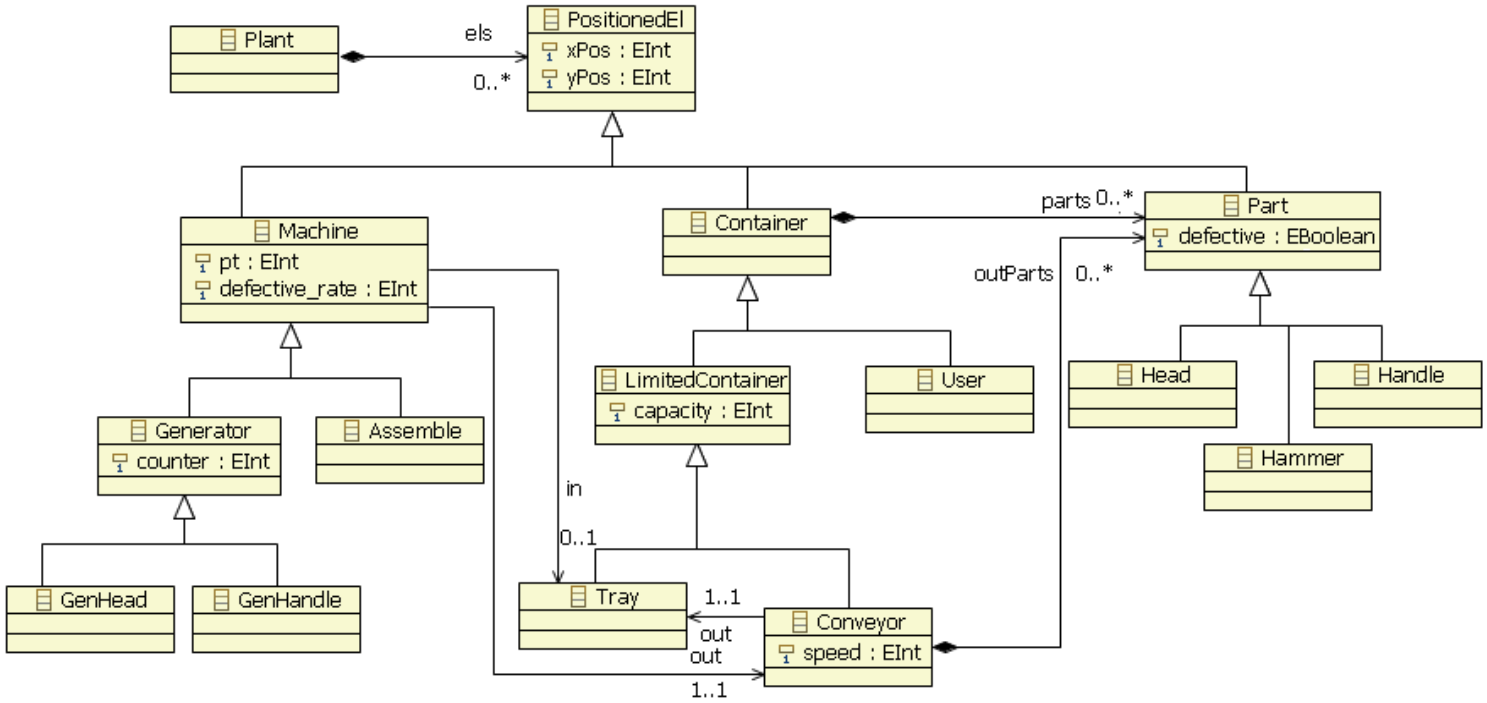
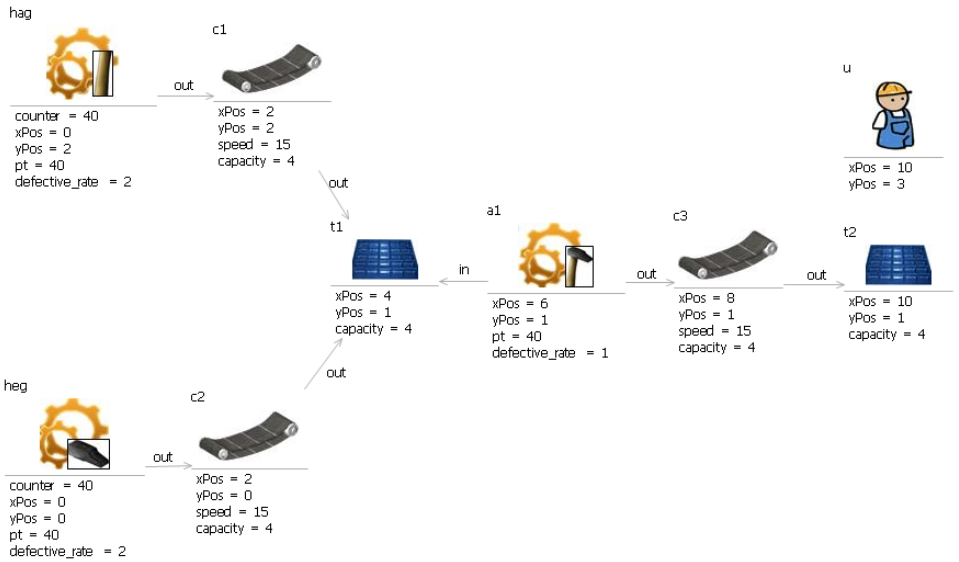
Describing Behaviour Models through Reusable, Multilevel, Coupled Model Transformations

Adrian Rutle **Fernando Macías** Francisco Durán
Roberto Rodríguez-Echeverría Uwe Wolter

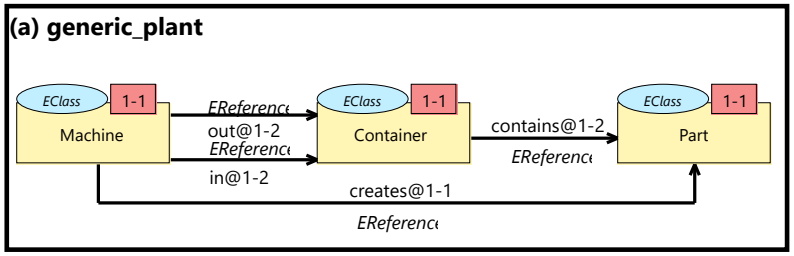
November 2, 2016

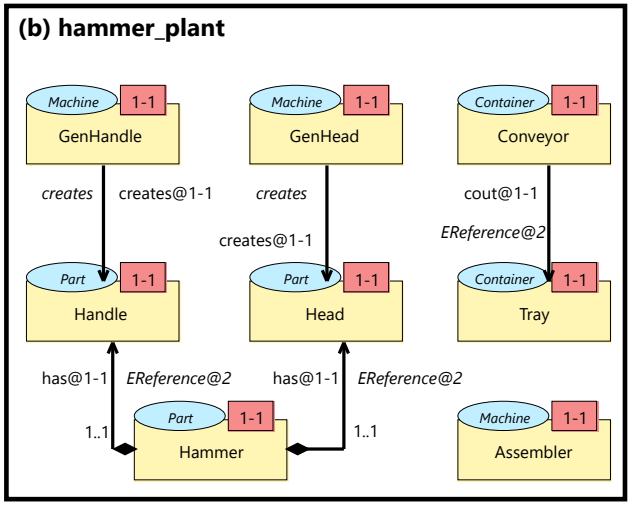
PLS example with two-level approach

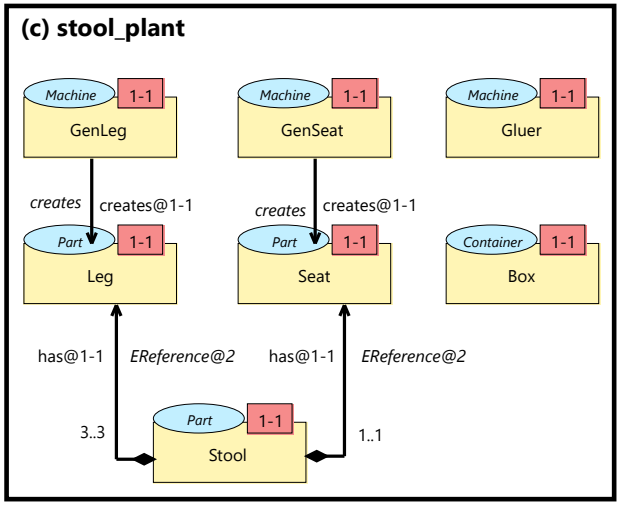


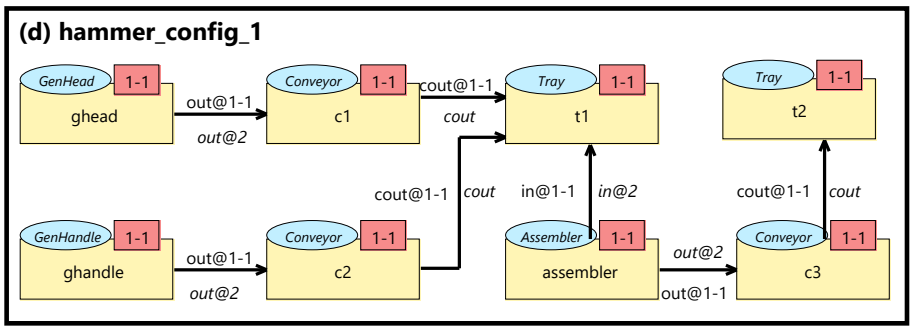


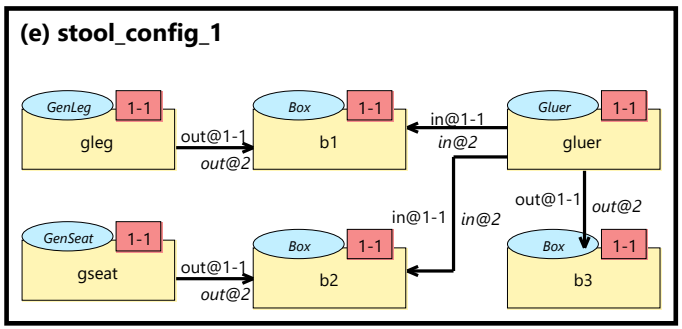
PLS example with multilevel approach



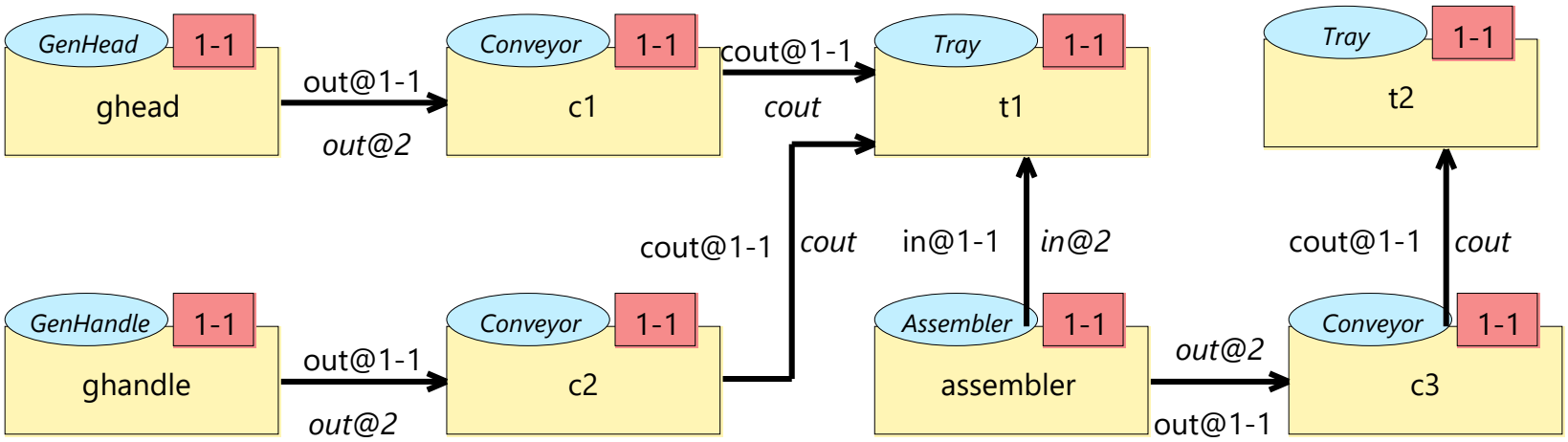








PLS example with multilevel approach



Defining semantics for behavioural models

- We use model transformations:
  - Inherently multilevel
- Commonality can be exploited for reusability
  - Across languages (horizontally)
  - Inside the same stack of languages (vertically)
- Generic if defined on higher levels
- Trade-off between genericity may lead to imprecision

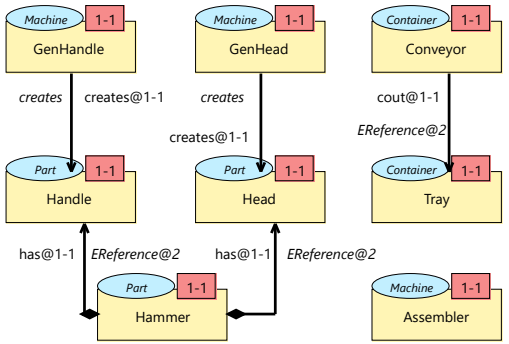
Defining semantics for behavioural models

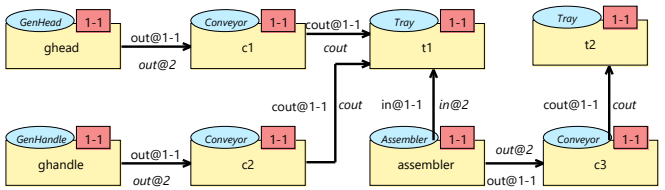
- We use model transformations:
  - Inherently multilevel
- Commonality can be exploited for reusability
  - Across languages (horizontally)
  - Inside the same stack of languages (vertically)
- Generic if defined on higher levels
- Trade-off between genericity may lead to imprecision

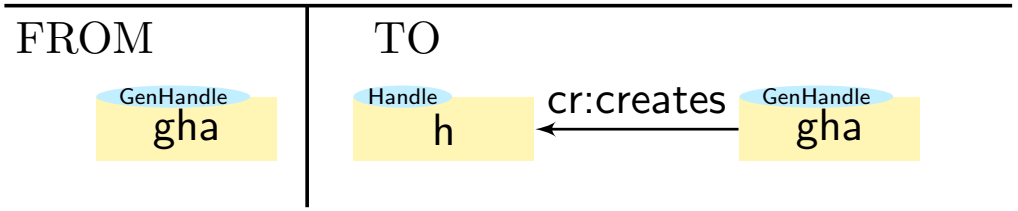
Our proposal: Multilevel transformations coupled with a meta-level: **precise and reusable**

Option 1: Two-level rule

- Create an instance of Handle for each instance of GenHandle

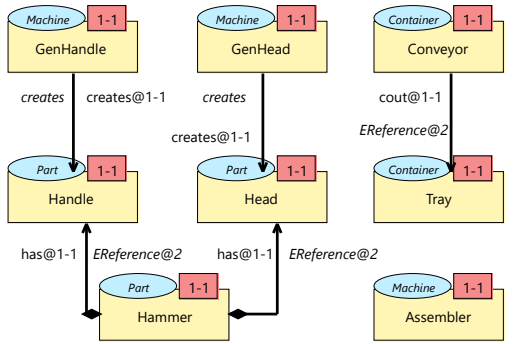


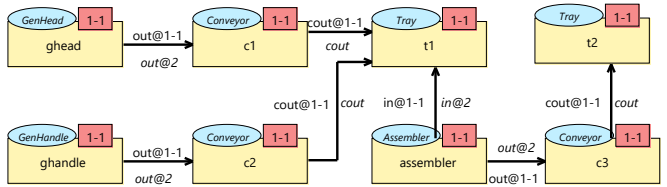


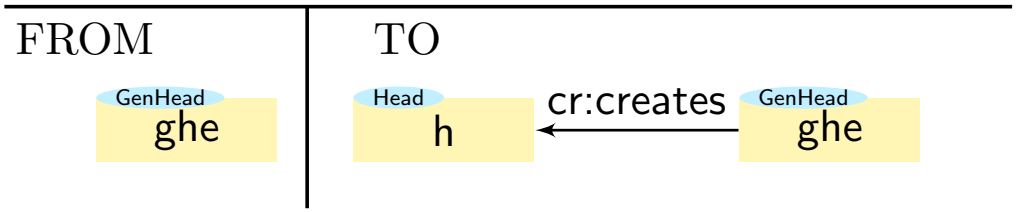


Option 1: Two-level rule

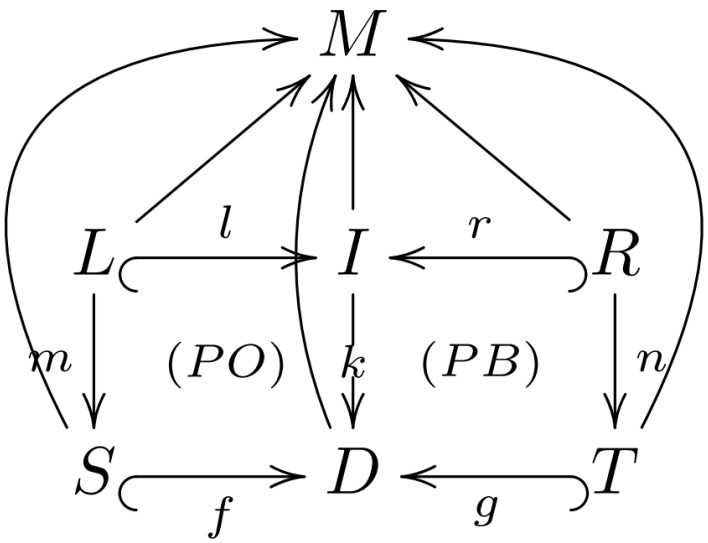
- Create an instance of Handle for each instance of GenHandle
- Create an instance of Head for each instance of GenHead

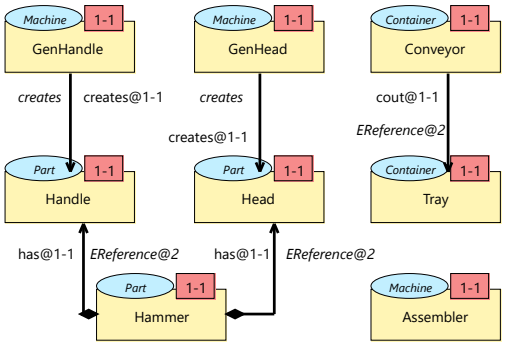


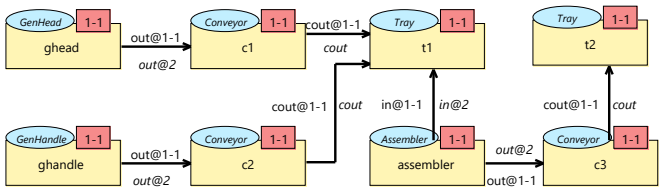


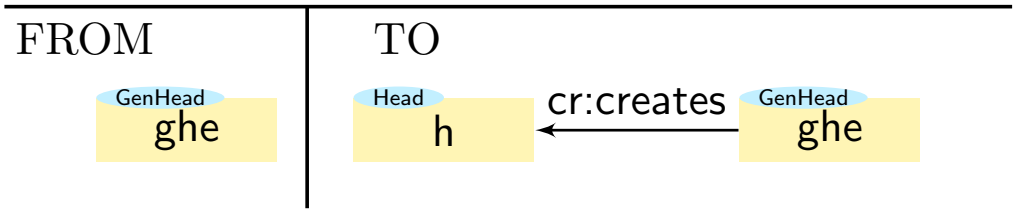


Option 1: Two-level rule





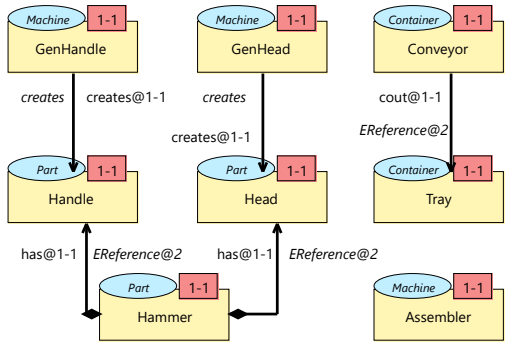


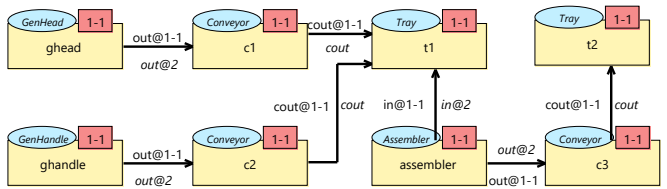


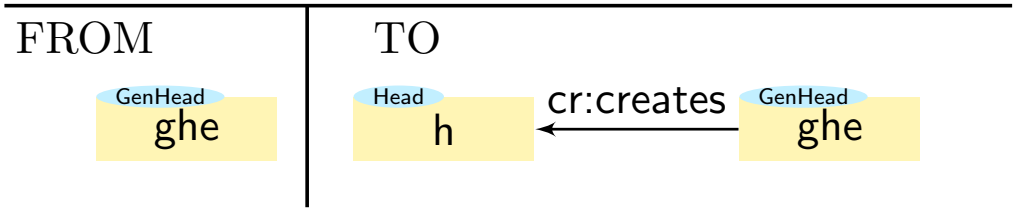
Option 1: Two-level rule

Problems

- × Too specific
- × Difficult to reuse
  - × Each machine needs a new rule
  - × Each language or hierarchy needs its set of similar rules
- × Leads to proliferation

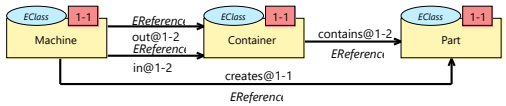


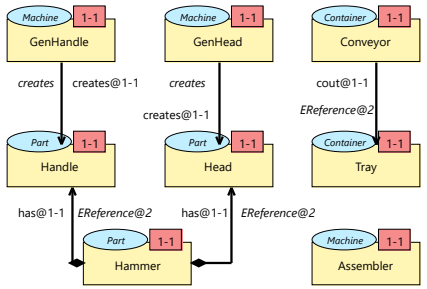


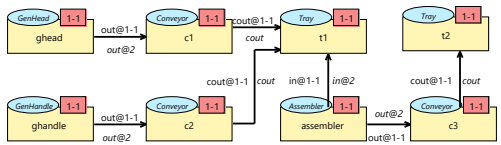


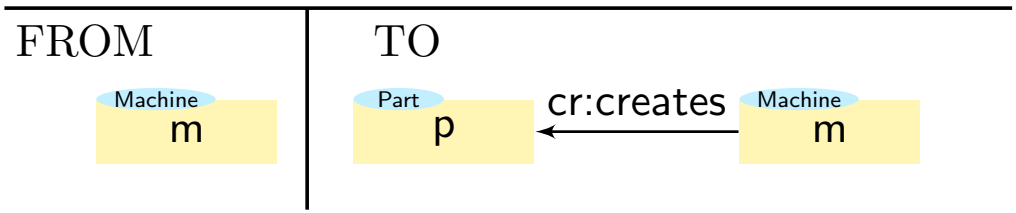
Option 2: Multilevel rule

- Create an instance of Part for each instance of Machine

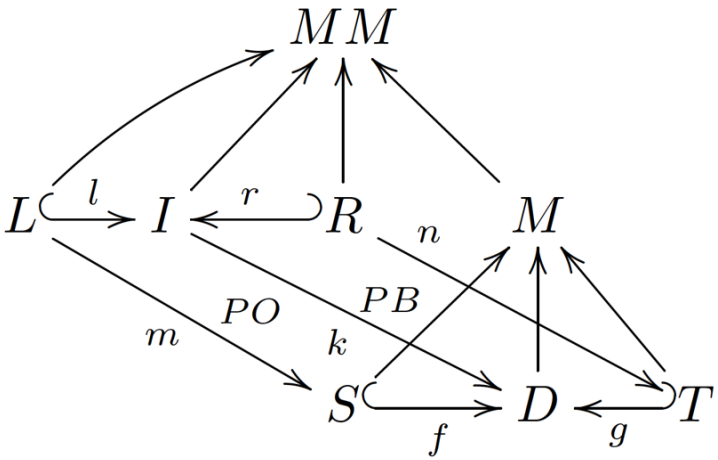


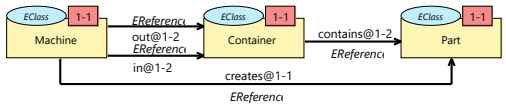


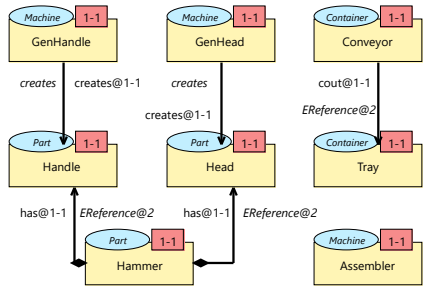


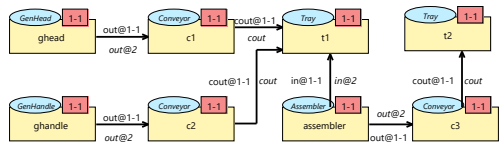


Option 2: Multilevel rule









FROM

Machine
m

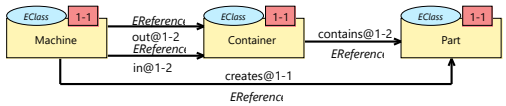
TO

Part
p

cr:creates

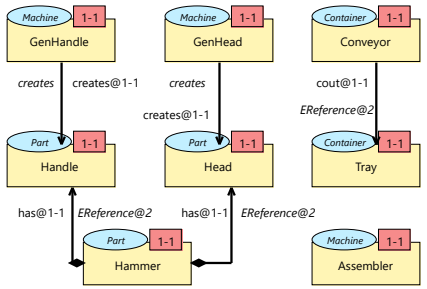
Machine
m

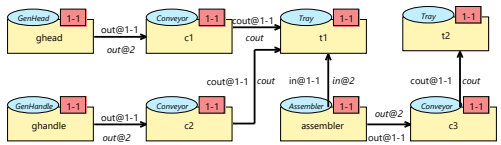
Option 2: Multilevel rule



Problems

- × Too generic
- × Not precise
  - × All machines will create parts
  - × All parts can be created directly
  - × Any machine can create any part





FROM

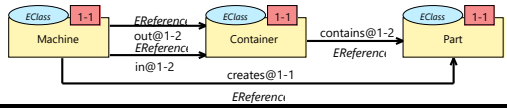
Machine
m

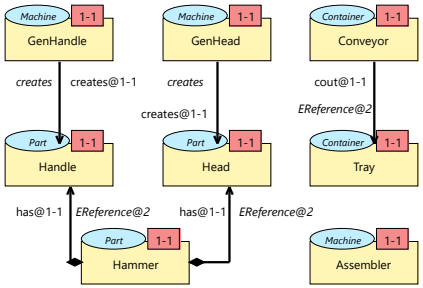
TO

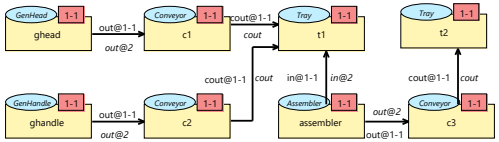
Part
p
cr:creates
Machine
m

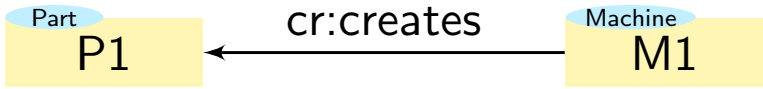
Option 3: Multilevel Coupled rule

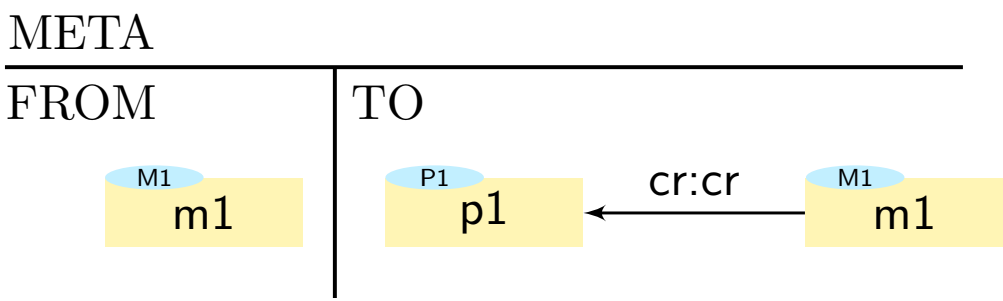
- Create an instance of each specific Part for each instance of Machine that generates that Part



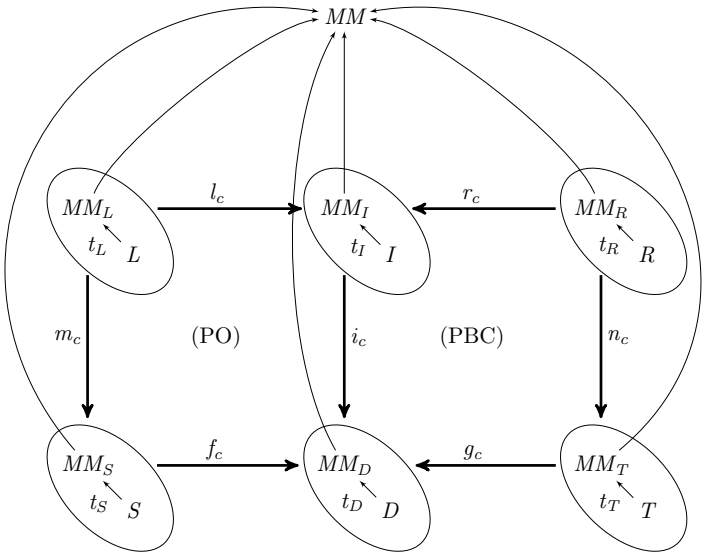


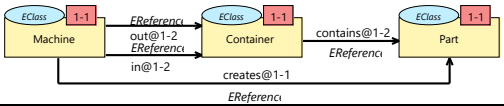


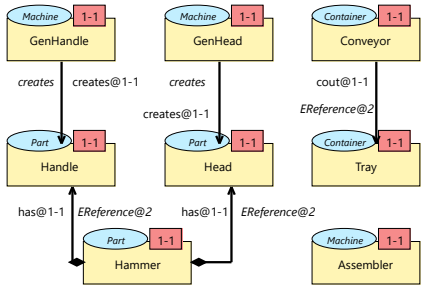


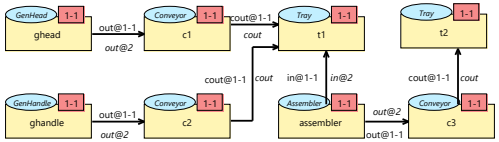


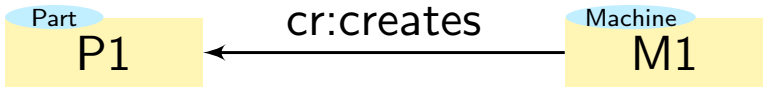
Option 3: Multilevel Coupled rule











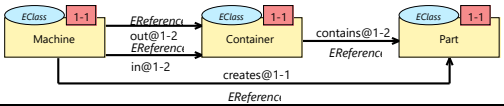
META

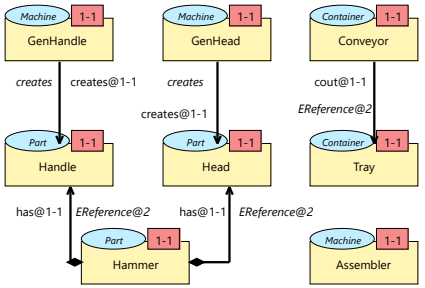
FROM	TO
M1 m1	P1 p1 cr:cr M1 m1

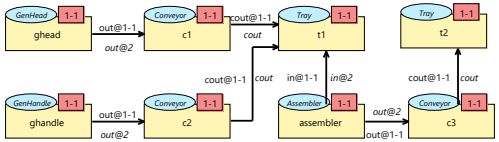
Option 3: Multilevel Coupled rule

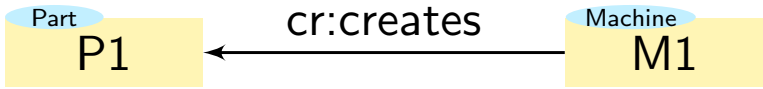
Advantages

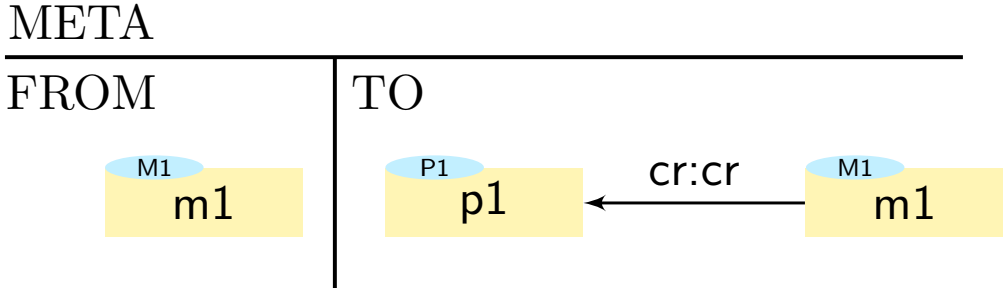
- ✓ Generic enough
  - ✓ Reusable in different configurations
  - ✓ Reusable in different languages
- ✓ Precise enough
  - ✓ Only the right machine creates a part





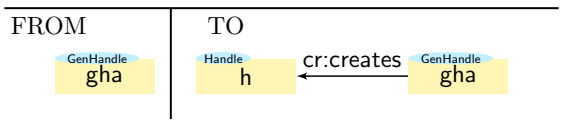


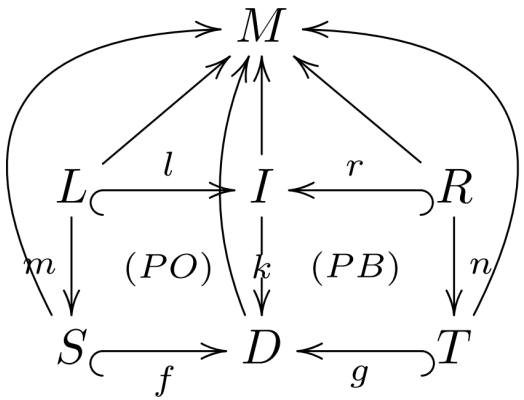




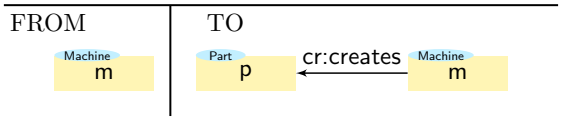
Comparison

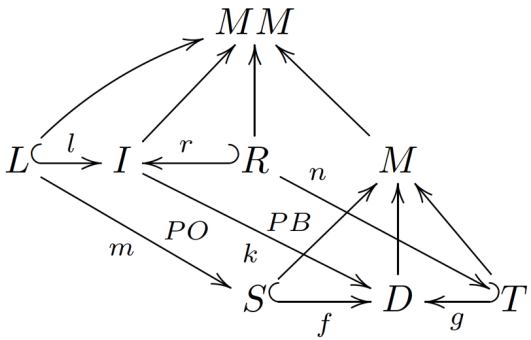
Two-level



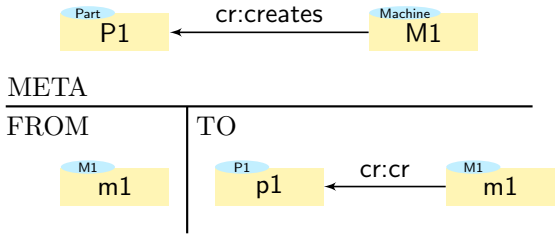


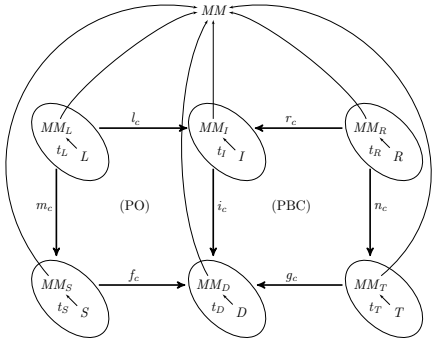
Multilevel





Coupled





Conclusions

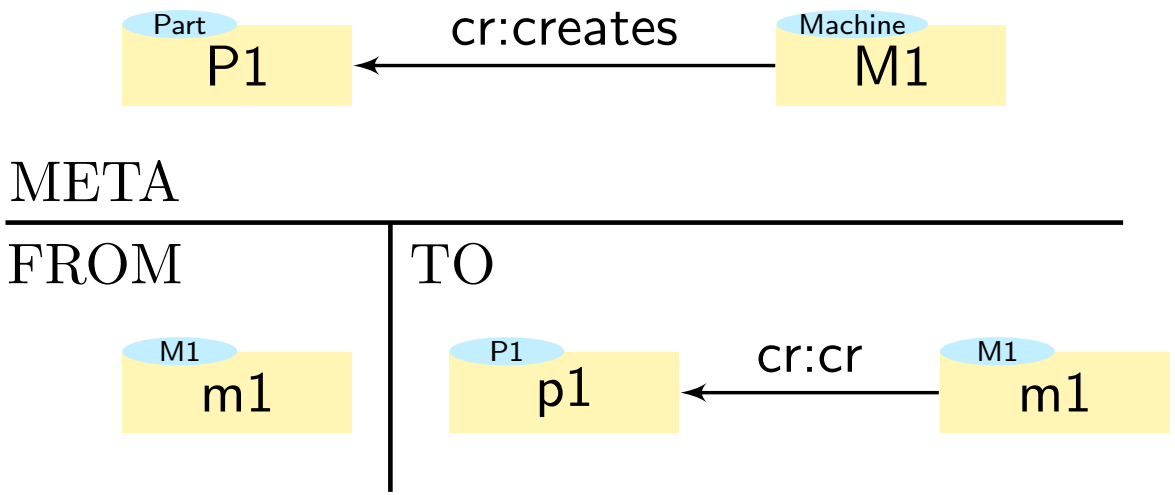
- Coupled model transformations exploit the advantages of multilevel metamodelling:
  - Reusability
  - Genericity
  - Precission
- Application to behavioural metamodelling natural and straightforward

Future Work

- Full formalization
- Expand horizontal and vertical flexibility
- Implement tool support

Extra slides

More Multilevel Coupled Rules – *CreatePart*

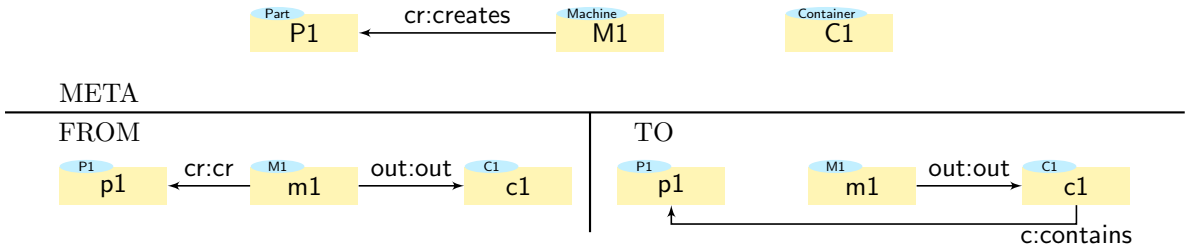



```
rule CreatePart {
  meta {
    P1: mm[0]!Part
    M1: mm[0]!Machine
    cr: mm[0]!Machine.creates
    [M1.cr = P1]
  }

  from {
    m1: M1
  }

  to {
    p1: P1
    m1: M1
    cr: cr
    [m1.cr = p1]
  }
}
```

More Multilevel Coupled Rules – *SendPartOut*

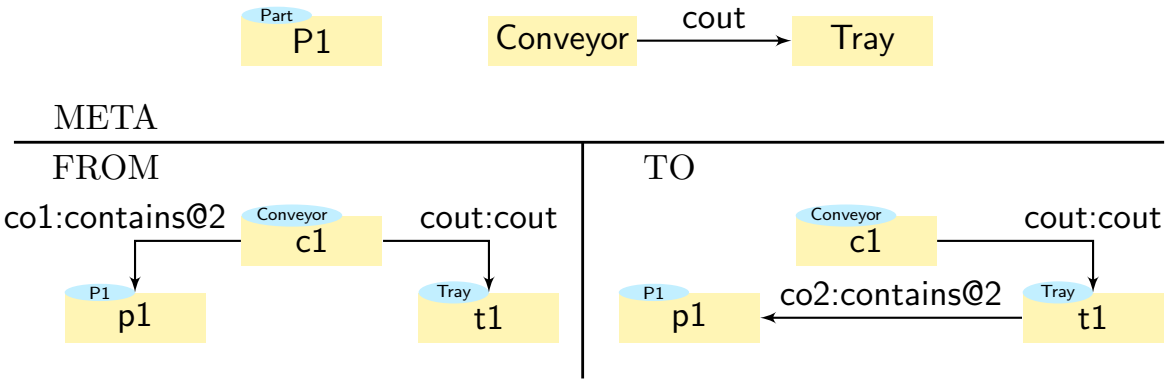



```
rule SendPartOut {
  meta {
    P1: mm[0]!Part
    M1: mm[0]!Machine
    C1: mm[0]!Container
    cr: mm[0]!Machine.creates
    out: $mm[0]!Machine.out
    contains: $mm[0]!Container.
          contains
    [M1.cr = P1]
  }

  from {
    p1: P1
    m1: M1
    c1: C1
    cr: cr
    out: out
    [m1.cr = p1]
    [m1.out = c1]
  }

  to {
     p1: P1
     m1: M1
     c1: C1
     out: out
     c: contains
     [m1.out = c1]
     [c1.c = p1]
  }
}
```

More Multilevel Coupled Rules – *TransferPart*

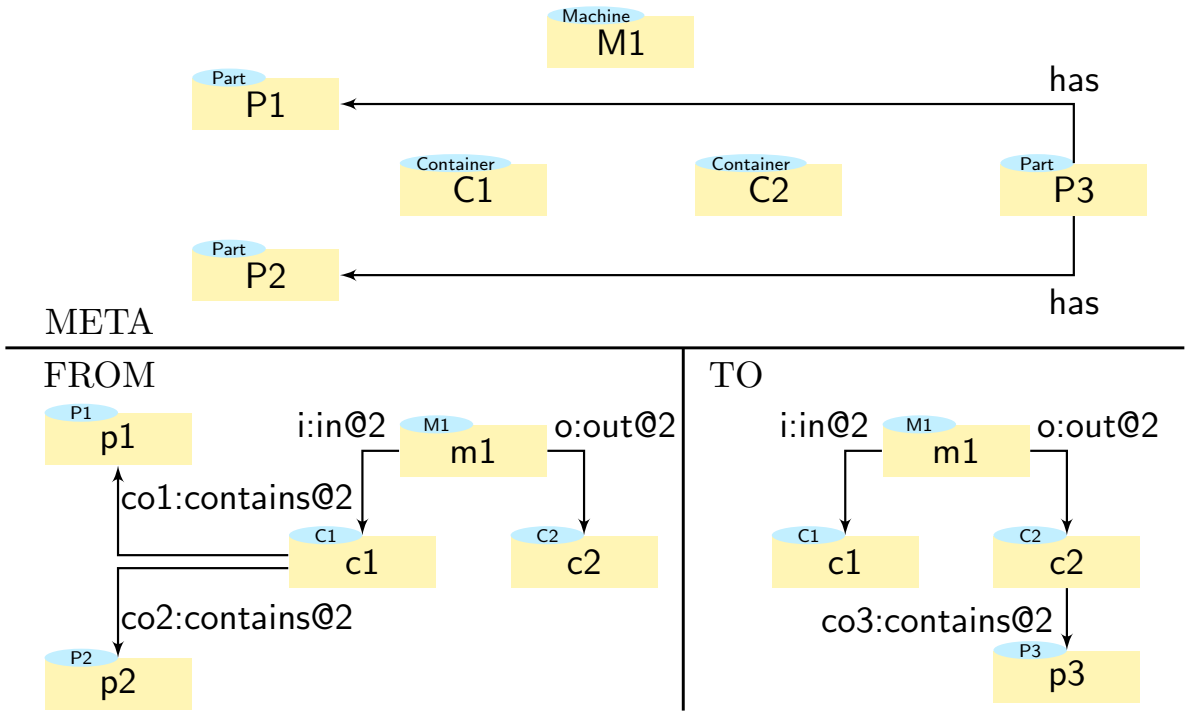



```
rule TransferPart {
  meta {
    P1: mm[0]!Part
    Conveyor: $mm[1]!Conveyor
    Tray: $mm[1]!Tray
    contains: $mm[0]!Container.
          contains
    cout: $mm[1]!Conveyor.cout
    [Conveyor.cout = Tray]
  }

  from {
    p1: P1
    t1: Tray
    c1: Conveyor
    cout: cout
    co1: contains
    [c1.co1 = p1]
    [c1.cout = t1]
  }

  to {
    p1: P1
    t1: T1
    c1: C1
    cout: cout
    co2: contains
    [t1.co2 = p1]
    [c1.cout = t1]
  }
}
```

More Multilevel Coupled Rules – *Assemble*




```
rule Assemble {
  meta {
    P1, P2, P3: mm[0]!Part
    C1, C2: mm[0]!Container
    M1: mm[0]!Machine
    c: $mm[0]!Container.contains
    in: $mm[0]!Machine.in
    out: $mm[0]!Machine.out
    has1, has2: mm[0]!Part.has
    [P3.has1 = P1]
    [P3.has2 = P2]
  }
  from {
    p1: P1, p2: P2
    c1: C1, c2: C2
    m1: M1
    i: in, o: out, co1: c, co2:
          c
    [c1.co1 = p1]
    [c1.co2 = p2]
    [m1.i = c1]
    [m1.out = c2]
  }
  to {
    p3: P3
    c1: C1, c2: C2
    m1: M1
    i: in, o: out, co3: c
    [c2.co3 = p3]
    [m1.i = c1] [m1.o = c2]
  }
}
```